UC Irvine
ICS Technical Reports

Title
Benchmarking for high-level synthesis

Permalink
https://escholarship.org/uc/item/9vw6210f

Authors
Gajski, Daniel D.
Dutt, Nikil D.

Publication Date
1992-06-30

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Z
699
C3
no. 92-69

Benchmarking for High-Level Synthesis

Daniel D. Gajski and Nikil D. Dutt

Technical Report #92-69
June 30, 1992

Notice: This Material
may be protected
by Copyright Law
(Title 17 U.S.C.)

Dept. of Computer Science
University of California, Irvine
Irvine, CA 92717
(714) 856-7219
duttics.uci.edu

Abstract

This paper discusses issues in benchmarking for synthesis, and suggests techniques for the comparison of benchmark descriptions, the synthesis tools used, as well as the synthesized designs finally generated. We propose a classification scheme for the assumptions made for the comparison of different synthesis tools, and present an Assumptions Chart *that can be used to visualize different benchmarks, tools and synthesis results. We illustrate application of this Assumptions Chart using synthesis experiments that were conducted on some sample High-Level Synthesis Workshop benchmarks.*

Notice: This Material
may be protected
by Copyright Law
(Title 17 U.S.C.)

Benchmarking for High-Level Synthesis

Daniel D. Gajski and Nikil D. Dutt
Department of Computer Science
University of California, Irvine
Irvine, CA 92717, U.S.A.

Abstract

This paper discusses issues in benchmarking for synthesis, and suggests techniques for the comparison of benchmark descriptions, the synthesis tools used, as well as the synthesized designs finally generated. We propose a classification scheme for the assumptions made for the comparison of different synthesis tools, and present an *Assumptions Chart* that can be used to visualize different benchmarks, tools and synthesis results. We illustrate application of this Assumptions Chart using synthesis experiments that were conducted on some sample High-Level Synthesis Workshop benchmarks.

1 Introduction

Silicon technology has reached the point where millions of transistors can be placed on a chip. The complexity of such chips has reached a level where human designers have difficulty in finding optimal solutions for their design implementations. This leads to a need for synthesis tools that can assist human designers in rapidly obtaining high-quality designs. Logic synthesis has been widely accepted and new tools for register-transfer and behavioral synthesis are emerging from universities and commercial CAD vendors.

The number of different techniques and synthesis algorithms has grown to the point where it is very difficult to compare them analytically or even experimentally using a set of benchmarks. Difficulty has also arisen in the selection of benchmarks and definition of the set of synthesis goals or design metrics by which to compare the synthesized benchmarks.

2 Benchmark Description Problems

The problem of comparing benchmarks for synthesis starts with the type and modeling style of the input descriptions for the benchmarks. These issues can be classified into realism, completeness, simulatability, level of detail, modeling style and the underlying design models.

Many synthesis benchmarks are unrealistic since they describe circuits that were never built or circuits that will never be be built. These benchmarks are also too simple to be taken seriously by the designers of commercial products.

The second problem with the benchmark descriptions is that different descriptions of the same circuit may not be easily comparable, due to the different levels of completeness in the benchmarks used for synthesis. Since complex benchmarks are difficult to synthesize, many benchmark descriptions are not complete: some less frequently used functions may be omitted from the description for the sake of simplicity, although these functions may actually take 90 % of the effort to synthesize. A simple example is the interrupt function that is omitted from several benchmark descriptions due to its difficulty in synthesis.

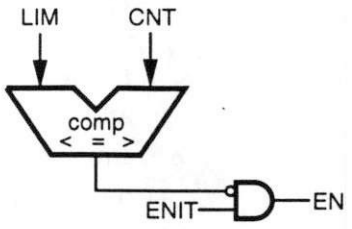
The third problem with benchmark descriptions is that several benchmarks are not simulatable. Simulatability is important since the input description is frequently modified or translated to suit different description languages (or even styles) at the input. Thus, tool developers must ensure that the benchmark's functionality is preserved (i.e., complete) after modification or translation. Hence a reasonable test vector set that exercises typical behaviors in terms of inputs and expected outputs, must accompany each benchmark. A minimal test vector set must include test vectors that will exercise at least each function in the design and toggle each input and output of the design. A more thorough test vector set should exercise each path through the benchmark design. Since the benchmark descriptions are behavioral and not structural, descriptions that define every path may be impossible.

The fourth problem is that two descriptions of the same design may have different amounts of detail. For instance, a behavioral description of a synchronous system may not have a clock explicitly defined and used within the design behavior; if more detail is added to the benchmark description that describes the clocking scheme and defines the clocks, the synthesis task becomes easier. In the extreme case, the implementation of the benchmark can be encoded within the description; synthesis then simply becomes a trivial task of decoding (or interpreting) the input description.

The fifth problem is that even complete descriptions of the same benchmark design that contain the same amount of detail may be different due to the use of a different modeling style [CaSt91] [LiGa91]. For example, Figure 1 shows two different descriptions of the same (complete) behavior that have the same amount of detail, but use different modeling styles. When asserted, the signal *ENIT* starts a counter by setting $EN = 1$. When the counter reaches its limit, the comparator asserts its output and sets $EN = 0$, which stops the counter. The first model treats *ENIT* as a level signal that stays asserted while the counter is counting. The second description uses the positive edge of the *ENIT* signal to set $EN = 1$ and the output of the comparator to set $EN = 0$. As shown in Figure 1, these two behaviors, although seemingly equivalent, result in two different implementations due to the differences in the modeling style. Since the modeler has chosen to use the positive edge of *ENIT* to indicate the moment when EN becomes equal to 1, the second implementation has an extra D-type flip-flop used to store the occurrence of the positive edge of the *ENIT* signal. The implementation shown in Figure 1(b) is correct but unnecessarily costly. This simple example shows that different modeling

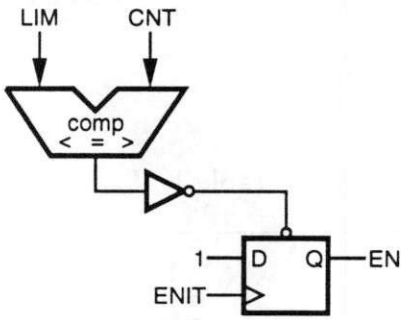
practices result in different designs and that complex synthesis algorithms are required for disambiguation of the design descriptions, making the task of benchmark comparison very difficult.

```
if CNT ≠ LIM then
    EN <= ENIT;
else
    EN <= '0';
end if;
```



(a)

```
if ENIT = '1' and not ENIT'stable then
    EN <= '1';
elseif CNT = LIM then
    EN <= '0';
end if;
```



(b)

Figure 1: Descriptions of an event with Different Modeling Styles: (a) Level Sensitive, (b) Edge Trigerred

The previous examples show the difficulty in comparing benchmarks that do not have similar modeling or description styles. An initial solution to the comparison of benchmark descriptions can be obtained by distinguishing the syntactic style, i.e., by looking at the variable typing, data operators and control constructs used. Some variable types are easy to synthesize (e.g., Boolean and integer variables), while arrayed variables and records require more effort. Furthermore, arrayed variables with constant indices are easier to synthesize, since data dependencies can easily be resolved during compilation. On the other hand, dependencies of arrayed variables with linear and non-linear indices are not easily recognized. Similarly, descriptions that only use Boolean operators are easily synthesizable, while floating point and complex number operations are not. Similarly, control constructs have different levels of difficulty in the range from straight-line code (SLC), simple ifs and loops, nested ifs and loops, to functions and procedures and special design oriented statements such as time delays. Table 1 shows a list of types, operations and control constructs ordered by the level of difficulty in synthesizing these constructs.

The final problem with the comparison of benchmark descriptions lies in the target architecture assumed in the input description. Input descriptions that assume different target architectures may describe designs with different information content, making

comparisons difficult. When the writing style of the description is combined with assumptions about the underlying design model, the task of benchmark comparison is worsened even further.

Types	*Operations*	*Control*
bit	Boolean	SLC
integer	add, subtract	simple if
real	shift	single loop
array (const)	multiply, divide	nested ifs
array (linear)	floating point ops	nested loops
complex vars	complex ops	functions
records	timing constructs	procedures
		concurrent processes
		resolution functions
		wait statements

Table 1. Types of Language Constructs

3 Synthesis Tool Problems

The synthesis of standard or ASIC chips from abstract benchmark descriptions involves many steps that span several design levels. Many of the synthesis steps are closely intertwined, and often result in the development of tightly coupled synthesis tools [GDWL92] [MiLD92]. Although synthesis tools in general incorporate several steps and span several design levels, none of the tools typically cover all the design levels and perform all the design steps within each level. In order to allow for comparison of different synthesis tools, we can first classify the tools by their levels of abstraction and the tasks they perform within each level.

System Synthesis refers to the task of converting an abstract benchmark description composed of a set of communicating processes into a set of chips, MCMs or PCBs. Typical synthesis tasks at the system level include grouping of variables, grouping of processes, grouping of communication channels, arbitration insertion, and partitioning of the description into chips, MCMs and boards.

High-level Synthesis takes a behavioral (algorithmic) description of a design and generates an RT-level design composed of a datapath and a controller. Typical high-level synthesis tasks include component selection, scheduling, allocation, binding and partitioning.

Sequential Synthesis takes the symbolic RT-level description generated by high-level synthesis, and performs state minimization, state encoding, interface synthesis, and memory synthesis to produce a controller at the logic level using generic gates.

Combinational Logic Synthesis applies logic minimization, technology mapping, and gate-level optimization (e.g., along critical paths) to realize the generic logic description using technology-specific library gates. Retiming can be applied to a logic circuit to improve the performance of the design.

Finally, *Physical Design* consists of tasks such as floorplanning, placement, routing, and transistor sizing to obtain the layout for the design.

One of the most difficult problems for the comparison of synthesis tools on benchmarks is to isolate one of the tasks from the others. Some synthesis tools perform multiple tasks in one algorithm, and different algorithms include different portions of each task. For example, some high-level synthesis algorithms perform scheduling and allocation together, while some others treat each task separately. Similarly, some logic synthesis algorithms perform technology mapping and transistor sizing together, while others may consider only technology mapping. Even at the layout level, different tools may perform any combination of placement, global routing and local routing. Benchmarking comparisons have to take into account the combination of different tasks, hence the decoupling of tasks for the purposes of comparison is a very difficult problem, even just at the layout level [Kozm91] [HiPr90].

The problem of comparing the results of different algorithms is exponentiated by the fact that different tools use different types of target architectures or design styles. A target architecture defines a design more precisely in terms of particular units, their parameters, and the connections among units. For example, a processor architecture would include the number of registers in the register file, the number of buses in the datapath, the number of pipeline stages, etc. The design style refers to the principal qualitative features of a design, such as prioritized interrupt, instruction buffer, bus-oriented datapath, etc.

Architectural models can be described for each of the abstraction levels defined earlier. System synthesis uses an architectural model composed of communicating Finite State Machines with Datapaths (FSMDs) [GDWL92], composed as shared-memory or message-passing multiprocessors. High-level synthesis uses the FSMD as an architectural model, with microarchitectural components as the building blocks. Sequential and combinational logic synthesis tools use an FSM or combinatorial logic model.

Furthermore, for each of these models, synthesis tools allow different amounts of engineering detail to be specified in addition to the functional description. Sample engineering characteristics include single-phase or multi-phase clocking, wire-or or tristate bussing, operation pipelining, control pipelining, and software pipelining. The addition of more engineering detail to the benchmarks makes the task of synthesis easier, since the benchmark has the engineering features explicitly encoded in the input description.

4 Technology Problems

Once the RTL or logic structure is synthesized from the behavioral descriptions, it must be manufactured using some IC implementation technology such as FPGAs, gate arrays, standard cells or custom silicon. Design quality metrics such as silicon area, yield, package

cost, clock rate, performance, power consumption, testability coverage, etc. depend upon the technology used (e.g., CMOS, GaAs), type of circuit (e.g., symmetric, dynamic logic, low power) and layout design goals.

In order to estimate design quality metrics during logic, RTL, and high-level synthesis, we must model the layout synthesis algorithms using parameters for the chosen technology implementation. These estimates of quality metrics need not necessarily be accurate as long as they provide high fidelity, that is, they allow tools or designers to consistently choose more optimal designs when two alternatives are compared.

These layout technology models or cost functions for RTL and High-Level Synthesis must account for all the technology and design tool parameters such as wire resistance and capacitance, buffer insertion, transistor sizing, wire lengths, floorplanning, placement, pad area and routing, logic minimization, technology mapping, state minimization, state encoding, and testability cost.

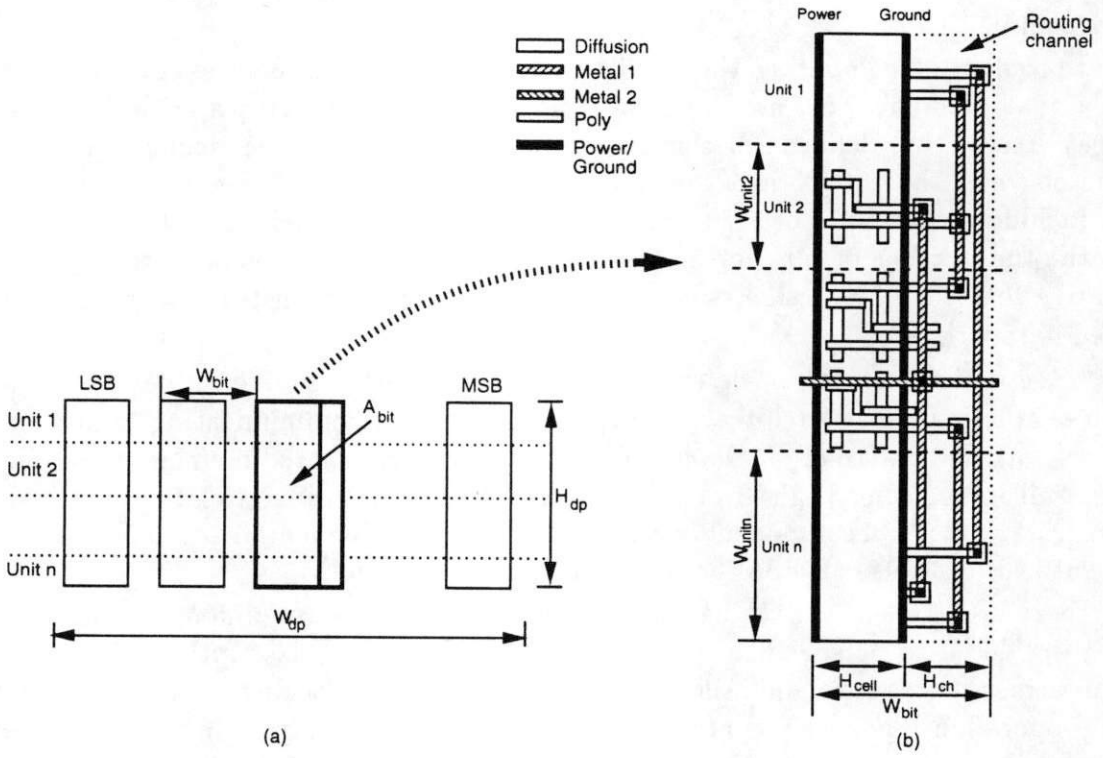


Figure 2: Standard Cell Datapath Stack: (a) Architecture, (b) Detailed Layout

We illustrate this process by giving an example of a layout area technology model for a datapath implemented using a bit-sliced standard cell architecture [WuCG91] [ChWG92] [RKGW92]. The datapath consists of a set of regularly structured RT components, such as ALUs, multiplexers, latches, drivers and shifters. Datapath layout is accomplished with a stack of functional and storage units that are placed one above the other. Each

unit consists of bit slices and all units have bit slices of the same width. However, bit slices in different units may be of different height. All bit slices are aligned starting with the least-significant bit (LSB) and distinct units are stacked on top of another. Thus, the stack grows horizontally when the bit width increases, and it grows vertically when the number of units increases (Figure 2(a)).

In this architecture, each bit slice of a unit is implemented with one row of connected standard cells as shown in Figure 2(a), with a bit slice of each unit consisting of of one or more standard cells (Figure 2(b)). P and N diffusion strips are placed vertically. Power and ground wires run vertically in the first metal layer. Control lines run over the standard cells in the second metal layer. Data lines are placed in the routing channel and run vertically in the first metal or the polysilicon layer. The connections between standard cells inside each bit slice is also placed in the routing channel.

To compute the height (H_{dp}) of a bit slice (Figure 2(a)), we observe that H_{dp} is proportional to the number of transistors in the bit slice. Each bit slice of a unit in Figure 2(b) can be computed as a product of the number of transistors ($tr(unit)$) and the transistor-pitch coefficient (α) in μm/transistor. α is obtained by averaging the ratio of cell width and the number of transistors per cell over all units in the library. Thus,

$$W_{unit} = \alpha \times tr(unit). \tag{1}$$

Consequently, the height of the bit-sliced stack of n units is

$$H_{dp} = \sum_{i=1}^{n} W_{unit_i} = \alpha \times (\sum_{i=1}^{n} tr(unit_i)). \tag{2}$$

The Equation 2 will hold for standard-cell architecture even if each bit slice is implemented with two or more rows of standard cells. Obviously, a different coefficient α' must be used in that case. Thus, the height of the bit-sliced stack of n units with an m-row implementation is

$$H_{dp} = \alpha' \times (\sum_{i=1}^{n} tr(unit_i))/m. \tag{3}$$

An estimate for the required number of tracks in each bit slice can be obtained only after the position of each unit in the bit slice is determined. A fast algorithm with pseudo linear time complexity, such as the min-cut algorithm can be used for this purpose. The required number of tracks can be estimated by the maximal density which is defined as the maximum number of connections across any cut perpendicular to the channel. A better estimate can be obtained by using some simple routing algorithms, such as the left-edge algorithm. Thus, the datapath area (A_{dp}) can be calculated as a product of the number of bits (b_w) and the area of one bit slice, i.e.,

$$A_{dp} = b_w \times H_{dp} \times (H_{cell} + H_{ch}). \tag{4}$$

The Equation 4 gives an upper bound on the datapath area. The bound is proportional to the product of the number of transistors and the number of routing tracks. The

number of transistors can be approximated from the Boolean expressions describing each unit slice or counted from its schematic. The number of tracks can be approximated by the track density after a linear placement. Better estimates can be achieved with algorithms of higher complexity. Since the number of components in the datapath is small, those more accurate estimates are not necessarily computationally intensive.

Such layout models need to be developed for use in conjunction with high-level synthesis tools, so that we can effectively compare the synthesis tools at higher levels using the same layout models. We also have to model particular algorithms used within the synthesis tools for comparison. For example, logic minimization must be accounted for when converting Boolean equations into transistors.

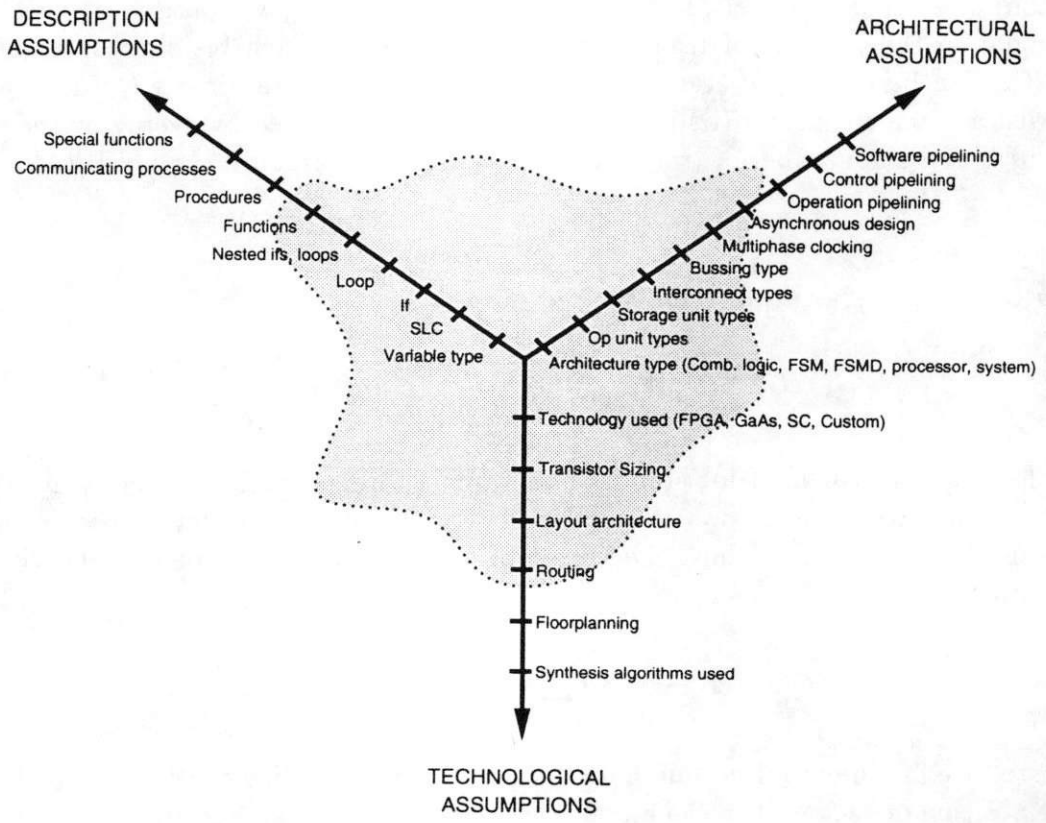


Figure 3: The Assumptions Chart for Benchmarking Comparison.

5 Solution

We saw that the design process consists of several tasks that use multiple algorithms for different design models and different design goals. Therefore when we want to compare or evaluate the performance of a design task or an algorithm with respect to a benchmark, we must indicate the assumptions made and models for the following three characteristics:

- Description Language Constructs
- Target Architecture
- Cost Function (Technology)

Each of these three characteristics can be combined into a Y-Chart as shown in Figure 3. This chart can be used to compare the assumptions made in the input description, the synthesis tool being tested, and the technology cost functions used. The *Description Assumptions* axis in Figure 3 indicates the descriptive level of difficulty in terms of syntactic features of the input language. The *Architectural Assumptions* axis shows the design model as well as the engineering features assumed by the synthesis tool. The *Technological Assumptions* axis shows the layout model or the technological cost functions used by synthesis tools. The comparative evaluation of synthesis tools on a set of benchmarks only makes sense when the tools make some set of assumptions on all the three axes, as represented by the shaded area on the assumptions chart in Figure 3. Without such an equivalence, it is hard (and perhaps meaningless) to embark on comparative evaluation of different synthesis tools' performance on a set of benchmarks.

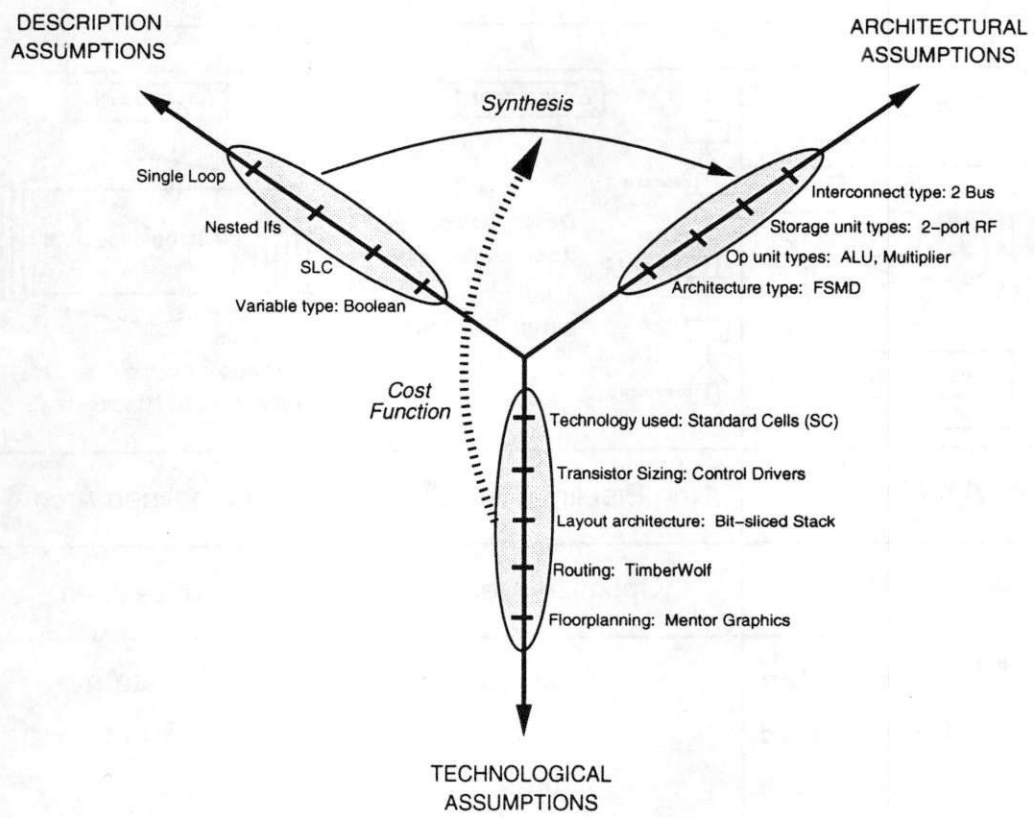


Figure 4: Application of The Assumptions Chart to Benchmarking.

As an example, Figure 4 shows that we will compare benchmarks that use only Boolean type variables, and *if* and *loop* control constructs to describe the FSMD architecture with 2-port register files, ALUs and multipliers using two buses as interconnection. The quality metrics are incorporated as a set of cost functions for synthesis and are based on a standard cell architecture with a bit-sliced datapath using cells with different transistor sizes to drive control lines across the datapath using Timberwolf for the routing of standard cells.

6 Experiments

In this section, we demonstrate the use of the Assumptions Chart to compare the results of synthesis on some standard High-Level Synthesis Benchmarks taken from [DuRa92]. We examine synthesis steps at various design levels and observe that the results of synthesis can vary substantially, based on the assumptions made with respect to the input description, the target architecture, the layout style (technological factors), as well as the cost functions used to drive the synthesis tools. As mentioned in the previous section, comparison of different synthesis tools only makes sense when the tools make equivalent (or similar) assumptions with respect to each axis of the Assumptions Chart.

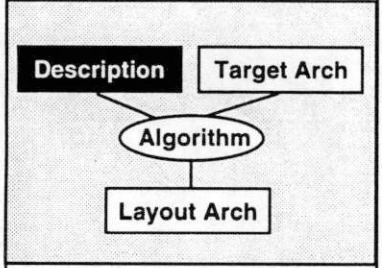
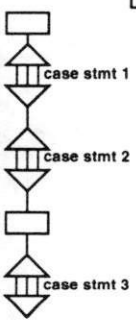
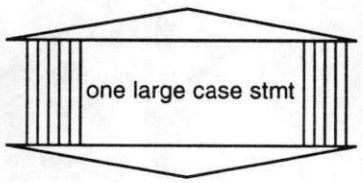
Am 2901	Experiment 1	Experiment 2
	 *Description contains 3 sequential case statements each having 8 options*	 *Description contains one case statement with 512 options*
Target Architecture	Non Pipelined Arch.	Non Pipelined Arch.
Cost Function	Optimize Area	Optimize Area
Results **Num. States**	**6 states**	**1 state**
Clock Period	**130 ns**	**280 ns**
Execution Time	**780 ns**	**280 ns**

Figure 5: Effect of Different Description Styles on Synthesis Results

6.1 Effect of Varying Input Description Styles

We first describe a sample experiment designed to investigate the effect of varying the input description style on the high-level synthesis task of scheduling using the TBS algorithm [RaGa91] in the VHDL Synthesis System (VSS) [LiGa88]. The design chosen for the experiment is the Am2901 benchmark, which is a bit-slice ALU controlled by a 9-bit word, partitioned into segments of three each for the source, operations and the destination of the ALU. The abstract behavior of the Am2901 can be described in two different ways. The first description style uses a set of three sequential case statements to check for the source, operations and destinations of the ALU (*Experiment 1* in Figure 5), while the second description style uses a single large case statement to test all the control lines simultaneously (*Experiment 2* in Figure 5).

Note that aside from the difference in input descriptions, all other parameters on the Assumptions Chart were identical: a non-pipelined architecture, an identical cost function for minimizing area, and an identical list scheduler. In spite of using the *same* scheduling algorithm for both the descriptions, the row labeled *Results* in Figure 5 shows that we get substantially different performance characteristics for the two descriptions.

The reason for the difference in the number of states is the algorithm used for scheduling. This algorithm schedules only across basic blocks and assigns a new control step for every *if* statement. Therefore, the description with more *if* statements is scheduled into more states, but results in a shorter clock period. On the other hand, when the description was flattened, the scheduler obtained fewer states with a longer clock cycle, but shorter overall execution time. However, writing a description with 512 case options is obviously painful; such descriptions will generally not be written and hence will not lead to synthesis tool acceptance.

6.2 Effect of Varying Target Architectures

Next, we describe a sample experiment designed to investigate the effect of varying the target architecture of the synthesized design, corresponding to different points on the *Architectural Assumptions* axis of Figure 3.

We start with the same VHDL behavioral (process) description of an industrial counter benchmark, and use the same area cost function for driving the scheduler and allocator, but target different architectures, as shown in Figure 6. The architecture in *Experiment 1* has no pipelining, while *Experiment 2* has a pipelined architecture with control and status registers.

The results of this experiment are almost obvious: the area of the synthesized design with a pipelined architecture (*Experiment 2*) is larger than that of the non-pipelined design (*Experiment 1*) by 41%, but the clock cycle and execution time on the worst-case path has improved for the pipelined architecture by 22%. However, note that this improvement was possible only because our scheduling algorithm was able to adapt to given architectural features such as control pipelining [RaGa92]. If the scheduling algorithm was written for only one architectural style, this comparison would not be possible.

6.3 Effect of Varying Layout Styles

We now describe a sample experiment designed to investigate the effect of varying the layout style for the synthesized design, corresponding to different points on the *Technological Assumptions* axis of Figure 3.

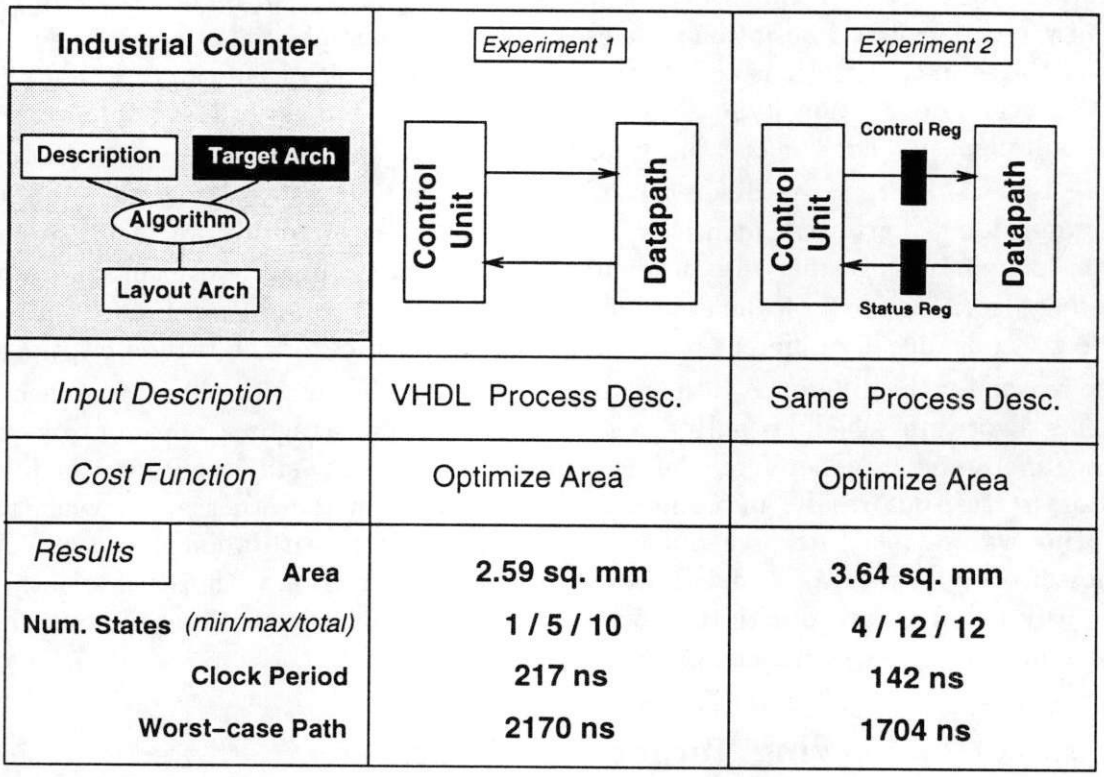


Figure 6: Effect of Different Target Architectures on Synthesis Results

In this experiment, we took the datapaths of several scheduled and allocated designs (17, 19, 21 and 28-step designs) for the Fifth Order Wave Elliptic Filter benchmark (i.e., the datapath output of high-level synthesis), and attempted to see if these designs consistently yielded the best and worst layouts across two different layout styles: bit-sliced stack and standard cells [RKGW92].

Figure 7 graphically shows the results of laying out the filter datapaths using a bit-sliced stack approach and a standard cell approach. It is interesting to note the relative shapes of the two curves in Figure 7. Although the 28-step filter design yielded minimum areas for the bit-sliced stack and the standard cell layout architectures, there is substantial variation in the relative ordering of the other filter designs. For example, the 17-step filter design yields the second-most area efficient design for the bit-sliced stack layout architecture, while yielding the *worst* area for the standard cell architecture.

This experiment clearly indicates the need to have well defined layout architectures and tools so that the outputs of high-level synthesis can be appropriately compared.

Furthermore, it emphasizes the importance of maintaining equivalent models on each axis of the Assumptions Chart when comparing different synthesis tools.

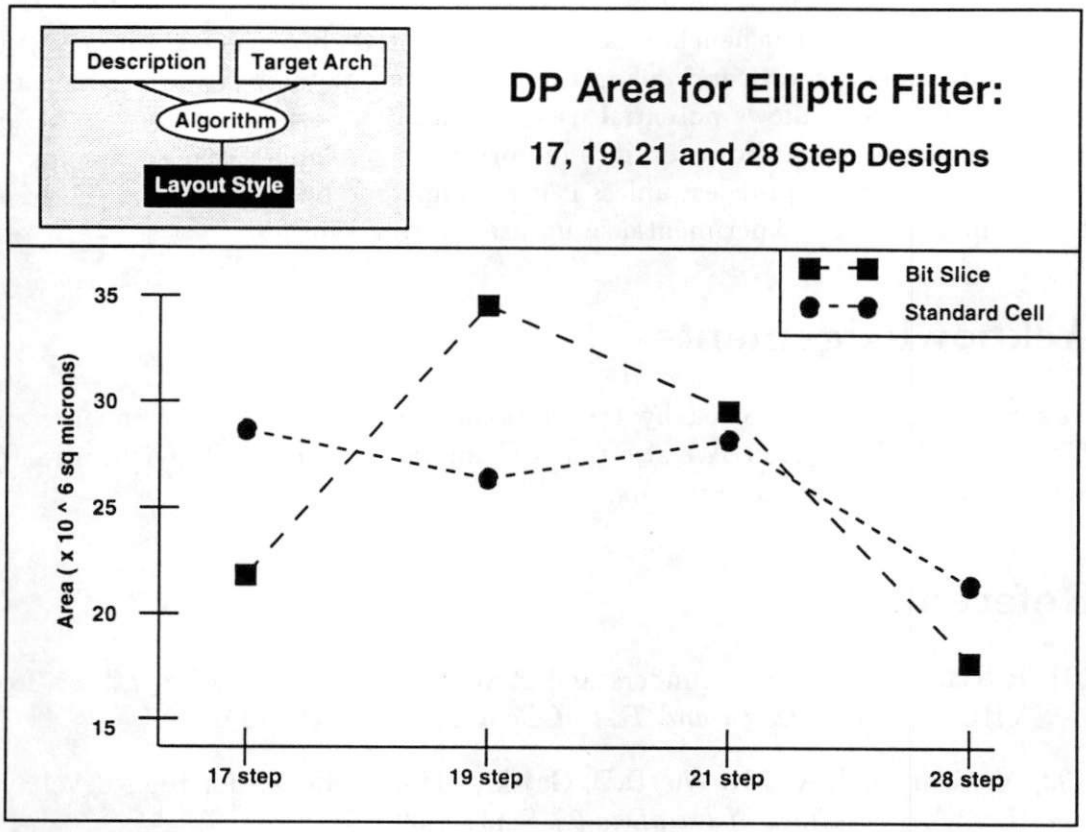


Figure 7: Effect of Different Layout Styles on Synthesis Results

7 Conclusion

In this paper, we described the issues involved in the benchmarking of synthesis approaches and tools and the problems faced in the interpretation and comparison of synthesis results. The reporting of synthesis results is meaningful only when performed in a controlled environment, where there is a clear definition of exactly *what* is being synthesized or compared, and under exactly which assumptions the synthesis is performed. In particular, the assumptions made in describing the input to synthesis, the architectural and layout models used, and the specific tasks performed by the synthesis task need to be *explicitly stated* in order to avoid ambiguities in the interpretation of synthesis results.

To ease the task of benchmarking and synthesis tool comparison, we presented a classification of assumptions for the comparison of different synthesis results and described an Assumptions Chart for visualizing the assumptions made by the synthesis results being

compared. We presented the results of some experiments that illustrated the method for comparison using proper description, architectural and implementation models for evaluating the quality of synthesis tools. These experiments also reinforce the importance of isolating the issues (e.g., synthesis task) being compared; if several of these tasks, models or assumptions are intermixed, then the results of the comparison become worthless.

In conclusion, we note that benchmarking is an expensive, but necessary process that permits tool developers to test and debug the capabilities of their synthesis, tools and systems. Furthermore, it allows potential users of such synthesis tools to evaluate and compare various approaches. However, it is important to remember that such comparative evaluation becomes meaningless unless it is accompanied by a careful analysis of the assumptions made and the experimental setup used during synthesis.

8 Acknowledgements

This work was partially supported by the National Science Foundation under grants MIP-8972851 and MIP-9009239. We also thank Champaka and Logie Ramachandran for their help in conducting the experiments.

9 References

[CaST91] R. Camposano, L.F. Saunders and R.M. Tabet, "High-Level Synthesis from VHDL," *IEEE Design and Test of Computers*, March, 1991.

[ChWG92] V. Chaiyakul, A.C-H. Wu, D.D. Gajski, "Timing Models for High-Level Synthesis,' *Proceedings of EuroDac-92*, Sept. 1992.

[DuRa92] N.D. Dutt and C. Ramachandran, "Benchmarks for the 1992 High-Level Synthesis Workshop," *ICS Technical Report 92-107*, University of California at Irvine, October 1992.

[GDWL92] D.D. Gajski, N.D. Dutt, A.C-H. Wu, and S. Lin, "High-Level Synthesis: Introduction to Chip and System Design," Kluwer Academic Publishers, 1992.

[GaDu92] D.D. Gajski and N.D. Dutt, "Benchmarking for High-Level Synthesis," *ICS Technical Report 92-69*, University of California at Irvine, June 1992.

[GaKu83] D.D. Gajski and R. Kuhn, "Guest Editors' Introduction: New VLSI Tools," *IEEE Computer*, vol. 16, no. 12, pp. 11-14, Dec. 1983.

[Kozm91] K. Kozminski, "Benchmarks for Layout Synthesis," *Proceedings of DAC-91*, June 1991.

[LiGa88] J.S. Lis and D.D. Gajski, "Synthesis from VHDL," *Proceedings of ICCD-88*, March 1988.

[LiGa91] J.S. Lis and D.D. Gajski, "Behavioral Synthesis from VHDL using Structured Modeling," *ICS Technical Report 91-05*, University of California at Irvine, January 1991.

[MiLD92] P. Michel, U. Lauther, P. Duzy, "The Synthesis Approach to Digital System Design," Kluwer Academic Publishers, 1992.

[HiPr90] D. Hill and B. Preas, "Benchmarks for Cell-Based Layout Systems," *Proceedings of DAC-90*, June 1990.

[PiBB92] M. Pilsl, S. Bhattacharya and F. Brglez, "Synthesizing Behavioral Benchmarks in VHDL into Standard Cell Layout," *Workshop Proceedings for the Sixth International Workshop on High-Level Synthesis*, Dana Point, CA, Nov. 1992.

[RaGa91] L. Ramachandran and D.D. Gajski, "An Algorithm for Component Selection in Performance Optimized Scheduling," *Proceedings of ICCAD-91*, Performance Optimized Scheduling," Nov. 1991.

[RaGa92] L. Ramachandran and D.D. Gajski, "Architectural Tradeoffs in the Synthesis of Pipelined Controls," *ICS Technical Report 92-49*, University of California at Irvine, May 1992.

[RKGW92] C. Ramachandran, F.J. Kurdahi, D.D. Gajski, A.C-H. Wu, V. Chaiyakul, "Accurate Layout Area and Delay Modeling for System Level Design," *Proceedings of ICCAD-92*, Nov. 1992.

[WuCG91] A.C-H. Wu, V. Chaiyakul, D.D. Gajski, "Layout-Area Models for High-Level Synthesis," *Proceedings of ICCAD-91*, Nov. 1991.

UC IRVINE LIBRARY
3 1970 01005 6288

MAY 27 1993